IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

First Named Inventor :	Lance Flake	
Appln. No.:	10/801,209	
Filed :	March 16, 2004	Group Art Unit: 2113
For :	METHODS AND STRUCTURE FOR ERROR CORRECTION IN PROCESSOR PIPELINE	Examiner: Wilson
Docket No.:	M142.12-0029	

BRIEF FOR APPELLANT

Commissioner for Patents P.O. Box 1450 Alexandria, VA 22313-1450	**Electronically Filed on October 29, 2007**

Sir:

This is an appeal from a Final Office Action dated July 13, 2007 in which claims 1-29 were rejected. Appellant respectfully submits that claims 1-29 are allowable, and requests that the Board reverse the rejection of claims 1-29 and find that claims 1-29 are in condition for allowance.

Contents

REAL PARTY IN INTEREST

Maxtor Corporation, a corporation of the state of Delaware, and having offices at 2452 Clover Basin Drive, Longmont, Colorado 80503, has acquired the entire right, title and interest in and to the invention, the application, and any and all patents to be obtained therefor, as set forth in the Assignment filed with the patent application and recorded on Reel 015107, Frame 0207.

RELATED APPEALS AND INTERFERENCES

There are no known related appeals or interferences which will directly affect or be directly affected by or have a bearing on the Board's decision in this appeal.

STATUS OF CLAIMS

Claims 1-29 were originally presented and have not been amended.

STATUS OF AMENDMENTS

There are no amendments that have been filed subsequent to the Final Office Action.

SUMMARY OF CLAIMED SUBJECT MATTER

1. Introduction

The present invention relates generally to pipelined processor operation in the presence of soft errors in memory and in particular relates to methods and structure for correcting

soft errors found in information (program instructions and/or data) read from an attached memory subsystem.

2. Brief Background

It is common practice in present day computer processor architectures to utilize pipelined architectures for improving performance. In particular, present-day processors utilize pipelined architectures in accessing memory subsystems. Programmed instructions and data are fetched and/or read from a memory in a pipelined fashion so as to optimize processor performance in fetching and executing instructions.

In general, "pipelined architecture" refers to an architecture of, for example, a computing processor in which various stages of the instruction and/or data memory read operations are substantially independently operable and thus may be overlapped with other processing stages within the processor to decode instructions, determine addresses of operands, reading operand data from memory and execute the instructions. The particular stages of a pipeline and the degree of independence of the pipeline stages are matters of design choice for the processor designer. By so overlapping various stages of processor instruction fetching and execution and related data reading and decoding, the processor may most effectively utilize available bandwidth from the memory subsystem. In other words, a next memory read or fetch operation may be initiated a while later stages of a previous fetch or read continue to completion. The overlapped operations of the various stages of memory access allow a new memory access as often as every clock period of the memory subsystem. Such pipelined architectures are well known to those of ordinary skill in the processor and memory architecture arts.

Memory subsystems coupled to such pipelined processors are rapidly evolving as measured by both capacity and performance. As memory speeds and densities increase, so to does the susceptibility of such memories to "soft errors." A "soft error," in general, is one that may be corrected by applying error detection and correction techniques and circuits. Such a soft error may be manifested as, for example, a flipped bit. In other words, a bit value previously stored in memory as a zero value is flipped to a one value, or vice versa, or otherwise erroneously read back. Such errors are most commonly caused by radiation of various types and frequencies that interfere with the persistent storage of data in the memory semiconductor circuits.

Processors may encounter such soft errors during instruction fetch or data read operations. When soft errors are encountered in a pipelined processor architecture, the overlapping sequencing of the pipeline stages may be stopped or disrupted (i.e., "stalled"). The pipeline stage may then need to be emptied and re-started to achieve the desired synchronization and overlap of stages of the memory interface pipeline. This stalling of the pipeline and the associated restarting thereof negatively impacts overall system performance.

However, the frequency of such soft errors is extremely rare as compared to the frequency of memory access due to the typical, error-free, pipelined instruction fetch and execution processes. Though somewhat infrequent, it remains an ongoing problem to reduce the frequency of encountering such soft errors or to correct them when detected.

One current solution to soft error problem in a memory subsystem is to add an error correcting code to all stored data and associated error detection and correction logic. Often such logic is embedded within the memory controller device used by the processor to interface with the memory subsystem. The error code

is stored along with data written to the memory. When data is read form memory, the associated error code is also read and may be used to detect the presence of an error and to correct the error. For example, one common error detecting and correcting code utilizes a six bit Hamming code associated with each of stored 32 bit value in the memory subsystem. Such a six bit Hamming code is capable of detecting multiple bit errors in the corresponding 32 bit stored value and is further capable of correcting any single bit error in the corresponding 32 bit stored value. During a write operation in such an enhanced, error correcting memory subsystem, the processor writes a new 32 bit value to the memory subsystem and the error correction logic associated with the memory subsystem generates the corresponding six bit Hamming code such that a 38 bit value is then stored in the identified memory subsystem location (32 bit value supplied by the processor and six bit Hamming code generated by the error correction features of the memory subsystem). During a read operation, the memory subsystem uses the retrieved 38 bits to produce a corrected 32 bit value even in the presence of any single bit error since the previous write operation.

However, such "in-line" correction typically slows the memory subsystem read performance for all memory accesses. In general, such error correcting techniques and logic add one or more "wait states" to every read or fetch operation. In view of the relative infrequency of such soft errors, imposing such a wait state penalty on every read or fetch operation is problematic and slows overall operation of the corresponding system.

Another presently known technique for reducing the frequency of soft errors is to provide a so called scrubbing engine. A scrubbing engine is an element operable as a background process that periodically reads each location in the memory, checks for errors using error correcting and detecting

codes and techniques, and corrects any detected errors by writing the corrected value back to the memory location from which the erroneous value was read. Such a scrubbing engine may impose one or more wait state delays on the memory subsystem performance but only when the scrubbing engine detects an error and attempts to write a corrected value back to the memory subsystem. While the scrubbing engine architecture can improve performance over an in-line error correcting memory subsystem design, the scrubbing engine architecture does not eliminate all possible soft errors but merely reduces the probability or frequency of such soft errors arising.

Yet another present solution for reducing system performance impact due to soft errors in a memory subsystem suggests that an in-line error correcting memory subsystem architecture may overlap the correction error detection in correction logic with presentation of the fetched data to the requesting processor. If it is later determined that the value returned from the memory subsystem to the requesting processor included a correctable error, the memory subsystem may in some manner interrupt the processor to cause invocation of appropriate error handling procedures within the processor. For example, the processor may be interrupted and provided with a corrected memory data value. Processors adapted for such a memory architecture may receive the interrupt signal and restart execution of the instruction corresponding to the erroneously fetched information. While such a process eliminates the wait state penalty of the above discussed in-line error correcting memory architectures, the re-execution of restart sequencing, per se, is complicated and can impose significant design tradeoffs within the processor pipeline architecture. Specifically, the processor pipeline design is complicated due to the lateness of the error signal relative to the processor's fetching and execution of the earlier, erroneous instruction or data.

All presently known solutions for reducing or eliminating soft errors in a memory subsystem incur certain performance penalties or complexities in a processor pipeline design. In-line error detection and correction architectures may add wait states to all memory accesses and thereby impose a significant cost on all memory read operations (regardless of whether an error is detected or corrected). Imposing such wait state delays on all memory accesses significantly diminishes overall system performance despite the relative infrequency of memory soft errors.

Failure to correct detected soft errors in-line (i.e., in real time as they are fetched or read) may impose other performance obstacles in that the pipelined fetch/read operations may be stalled when an error is detected and/or corrected thereby requiring restart of the pipeline architecture. Such pipeline architecture stalls and restarts cause other performance problems in the overall system and add design complexities. A processor pipeline stall and restart generally entails emptying the pipeline of multiple instruction and data read operations in various stages and then restarting the fetch/read operations so purged from the pipeline.

3. The Present Invention

Claims 1 and 17 are the only independent claims on appeal.

Independent claim 1 provides a processor (shown in Figure 1, and described, at least, on page 9, lines 23 - 33) having a memory interface comprising a multi-stage pipeline for fetching or reading information from a memory coupled to the processor. The pipeline includes a read stage (shown in Figure 1, and described, at least, on page 11, lines 16 - 17), a correction stage (shown in Figure 1, and described, at least, on page 11,

lines 17 - 22), and a utilization stage (shown in Figure 1, and described, at least, on page 11, lines 24 - 30). The read stage reads a unit of information from the memory. The correction stage corrects a soft error detected in a read unit of information. The utilization stage utilizes information in the corrected information.

Independent claim 17 provides a method for correcting soft errors in a pipelined processor coupled to a memory subsystem. The method comprises reading (shown in Figure 4, and described, at least, on page 17, lines 19 - 31) a unit of information from an attached memory in a read stage of the processor pipeline. The method also comprises correcting (shown in Figure 5, and described, at least, on page 18, lines 22 - 31) a soft error in the read information in a correction stage of the processor pipeline. The method also comprises utilizing (shown in Figure 6, and described, at least, on page 19, lines 14 - 28)the corrected information in a utilization stage of the processor pipeline.

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Whether claims 1-29 are patentable over U.S. Patent No. 5,604,753 A to Bauer et al. (hereinafter - Bauer).

Appellant respectfully submits that claims 1-29 are patentable over the reference, and requests that the Board find likewise and accordingly reverse the rejection of claims 1-29 and find these claims allowable.

ARGUMENT

1. Introduction: Claims 1-29 Should Be Allowed

With this appeal, Appellant respectfully requests that the Board reverse the rejection of claims 1-29 under 35 U.S.C.

102(b), based on failures of the reference to teach each and every limitation of the claims.

2. The Law of Anticipation

As set forth in Section One of the Final Office Action, 35 U.S.C. §102(b) provides that a person shall be entitled to a patent unless, "The invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States." Further, Appellant respectfully notes that the Federal Circuit has provided guidance with respect to anticipation. The Federal Circuit has held anticipation to be present, "If every limitation in a claim is found in a single prior art reference." See Nystrom v. Trex Co., 71 U.S.P.Q.2.d. 1241 (Fed. Cir. 2004). Appellant respectfully submits that each and every element of independent claims 1 and 17 is not found in the Bauer reference.

3.0 The rejection of claims 1-29 does not comply with the law of anticipation

3.1 Claims 1 and 17 are not anticipated by Bauer

Section Two of the Final Office Action indicated that independent claims 1 and 17, among others, were rejected under 35 U.S.C. §102(b) as being anticipated by Bauer.

Independent claim 1 provides, among other things, a processor having a memory interface comprising a multi-stage pipeline for fetching or reading information from a memory coupled to the processor. The pipeline is further defined to include, among other things, a correction stage to correct a soft error detected in a read unit of information. Similarly,

independent claim 17 provides a method for correcting soft errors in a pipeline processor coupled to a memory subsystem. The method includes, among other things, correcting a soft error in the read information in a correction stage of the processor pipeline. Appellant respectfully believes that the Final Office Action has not given proper effect to the claim language directed to the pipelined processor, as well as the correction stage.

3.1.1 Bauer does not provide a "multi-stage pipeline"

Section Two of the Final Office Action asserted, "As per claims 1, 17, Bauer et al. discloses a multi-stage pipeline for fetching or reading information from a memory coupled to the processor, see Figure 6, the pipeline including ... a correction stage to correct a soft error detected in a read unit of information in column 6, lines 17-34...."

As a threshold matter, it is important for proper meaning to be given to the term "pipeline" in the claim language. At least one resource reports that, in the context of computing, a "Pipeline" is a set of data processing elements connected in series, so that the output of one element is the input of the next one." (Emphasis Added). See Pipeline (computer), http://en.wikipeda.org/w/index/php?title=Pipeline_%28computer%29&oldid=105423028 (last visited February 15, 2007). Accordingly, in the current context, the recitation of the "pipeline" limitation requires multiple stages where the output of one stage is the input of a next one. This is important because Bauer, in providing a method and apparatus for performing error correction on data from an external memory (see Title), does so in a deliberately non-pipelined manner. For example, column 5, lines 55-59 provide,

> "In the present invention, the desired data returned is sent directly to execution core 101A for immediate use, while the error correction is performed in parallel by ECC logic 101C. Thus the present invention bypasses

> the data directly to the execution core 101A without error correction initially." (Emphasis supplied).

Section Thirty-Two of the Final Office Action appears to be responsive to Appellant's previous explanation that the teachings of Bauer are precisely the opposite of the pipeline features recited in claims 1 and 17. Specifically, Section Thirty-Two indicates that the Examiner respectfully disagrees. Further, Section Thirty-Two provides, "As disclosed in Figure 6, the execution core is function [sic] as a pipeline. The outputs and inputs are connected to one another as in accordance of a pipeline." Appellant vigorously asserts that Bauer simply does not teach or suggest the pipeline features recited in claims 1 and 17. Accordingly, Appellant respectfully believes that the teachings of Bauer are precisely the opposite of the pipeline features recited in claims 1 and 17.

3.1.2 Bauer does not provide a correction stage in a multi-stage pipeline.

The Final Office Action asserted that Appellant's "correction stage" is met by disclosure in Bauer at column 6, lines 17 - 34. However, the cited portion only speaks of the data path between the L2 cache memory and the processor. Specifically, the description therein refers to data correction block 204 and data correction block 216. However, there does not appear to be any indication that either data correction blocks 204 or 216 are anything that would be equivalent to a correction stage of a pipelined processor. Further still, there is simply no indication that block 204 and/or 216 are sub-components of execution core 503, asserted by Section Two of the Office Action as being the multi-stage pipeline for fetching or reading information from a memory coupled to the processor.

FIG. 6 and its related description describes execution core 503 of processor 402/412. However, the error correction of Bauer is done in parallel with the transmission of the uncorrected data to the execution core. See column 5, lines 55-59. Further, "the corrected data is then forwarded to the requesting unit during the next cycle. Also if an error is detected, the present invention [of Bauer] produces an indication to the device." See column 3, lines 3-5. (Emphasis is supplied). In distinct contrast, the stages recited in the independent claims each provide their respective outputs to the input of a next stage. As set forth on page 11 of Appellant's specification, "Read pipeline stage 106 applies the retrieved unit of information and associated error correction code to the soft error correction pipeline stage 108. Error correction pipeline stage 108 utilizes the error correcting code to detect the presence of an error in the unit of information read by the pipeline stage." Further as set forth on page 11, lines 20-26 of Appellant's specification,

> "The unit of information so read by read stage 106 and corrected by correction stage 108 may then be applied to decode pipeline stage 110. Decode pipeline stage 110 is operable to parse the corrected instruction/data information retrieved and corrected by stages 106 and 108 to decode the information to extract details regarding the particular instruction to be executed. Having so decoded an instruction and any associated operand data, execution (ALU) stage 112 is then operable to execute the corrected instruction and the data information."

Accordingly, embodiments of the invention recited in independent claims 1 and 17 do not later determine an error, and then interrupt the processor causing the "memory request [to] be purged from execution core 101A." as taught by Bauer, and described as prior art in the first full paragraph on page 4 of Appellant's specification. If Bauer functioned as asserted by the Final Office Action, the error correction would necessarily be

done in series with the other processing, and that is precisely the opposite of what Bauer teaches. Accordingly, Appellant respectfully submits that independent claims 1 and 17 are allowable over Bauer.

4.0 Conclusion

As set forth above, Appellant respectfully submits that each of independent claims 1 and 17 are allowable over the Bauer reference. Additionally, Appellant respectfully submits that dependent claims 2-16 and 18-29 are allowable over the Bauer reference by virtue of their dependency from allowable independent claims. Accordingly, Appellant respectfully requests that the Board reverse the decision of the Examiner and find that claims 1-29 are allowable.

The Director is authorized to charge any fee deficiency required by this paper or credit any overpayment to Deposit Account No. 23-1123.

Respectfully submitted,

WESTMAN, CHAMPLIN & KELLY, P.A.

By: 

Christopher R. Christenson, Reg. No. 42,413
Suite 1400, 900 Second Avenue South
Minneapolis, Minnesota 55402-3319
Phone: (612) 334-3222 Fax: (612) 334-3312

CRC:sew

Claims Appendix

Claims on appeal, as they currently stand:

1. A processor having a memory interface comprising:
a multi-stage pipeline for fetching or reading information from a memory coupled to the processor, the pipeline including:
a read stage to read a unit of information from the memory;
a correction stage to correct a soft error detected in a read unit of information; and
a utilization stage to utilize information in the corrected information.

2. The processor of claim 1
wherein the read stage comprises an instruction read stage to read a program instruction unit of information from the memory,
wherein the correction stage comprises an instruction correction stage to correct a detected soft error in the read instruction, and
wherein the utilization stage comprises an instruction decode stage to decode the corrected instruction unit of information.

3. The processor of claim 1
wherein the read stage comprises an instruction read stage to read a program instruction unit of information from the memory,
wherein the correction stage comprises an instruction correction stage to correct a detected soft error in

the read instruction, and

wherein the utilization stage comprises an instruction execution stage to execute the corrected instruction unit of information.

4. The processor of claim 1

wherein the read stage comprises a data read stage to read a data unit of information from the memory,

wherein the correction stage comprises a data correction stage to correct a detected soft error in the read data, and

wherein the decode stage comprises a data decode stage to decode the corrected data unit of information.

5. The processor of claim 1 wherein the read stage is adapted to read a previously stored unit of information from the memory and an associated error correction code previously stored in the memory.

6. The processor of claim 5 wherein the previously stored unit of information is 32 bits and the previously stored error correction code is a 6 bit Hamming code.

7. The processor of claim 1 wherein the read stage and the correction stage are both operable within a single cycle of the attached memory.

8. The processor of claim 1 further comprising:

control logic to enable and disable operation of the correction stage.

9. The processor of claim 1 further comprising:

correction logic to write the corrected data back to the

memory; and

notification logic coupled to the correction logic to signal correction the correction logic that corrected data is available.

10. The processor of claim 9 wherein the notification logic includes:

error storage for storing the address of the corrected data in the memory.

11. The processor of claim 10 wherein the error storage further includes:

error data storage for storing the erroneous value read from the memory.

12. The processor of claim 1 wherein the multi-stage pipeline further comprises:

a write correction stage to write corrected data back to the memory.

13. The processor of claim 9 wherein the correction logic is implemented as programmed instructions to be executed by the processor.

14. The processor of claim 13 wherein the notification logic is adapted to generate an interrupt signal in the processor and wherein the correction logic is executed in response to detection of the interrupt signal.

15. The processor of claim 1 wherein the correction stage is operable within a single cycle of the attached memory.

16. The processor of claim 1 wherein correction of a soft error

requires more than a single cycle of the attached memory and wherein the pipeline further includes:

multiple correction stages to correct a soft error detected in a read unit of information.

17. A method for correcting soft errors in a pipelined processor coupled to a memory subsystem, the method comprising:

reading a unit of information from an attached memory in a read stage of the processor pipeline;

correcting a soft error in the read information in a correction stage of the processor pipeline; and

utilizing the corrected information in a utilization stage of the processor pipeline.

18. The method of claim 17 wherein the step of correcting comprises:

correcting the read information in multiple correction stages of the processor pipeline.

19. The method of claim 17 wherein the steps of correcting and utilizing are performed within a single memory cycle of an attached memory system.

20. The method of claim 17 further comprising:

selectively disabling operation of the correction stage.

21. The method of claim 17 wherein the step of reading comprises:

reading a unit of data; and

reading a corresponding error correcting code previously stored with the unit of data.

22. The method of claim 21 wherein the step of reading a unit of

data comprises reading a 32 bit data value, and wherein the step of reading a corresponding error correcting code comprises reading a 6 bit Hamming code.

23. The method of claim 17 further comprising:
storing information regarding a soft error corrected in the step of correcting.

24. The method of claim 23 wherein the step of storing comprises:
saving an address value of the location that provided the corrected soft error.

25. The method of claim 24 wherein the step of storing further comprises:
saving an erroneous data value that provided the corrected soft error.

26. The method of claim 17 further comprising:
notifying the processor that a soft error was corrected.

27. The method of claim 26 further comprising:
executing instructions in the processor to write the corrected information into the memory.

28. The method of claim 26 wherein the step of notifying comprises:
interrupting the processor to signal correction of a soft error.

29. The method of claim 17 further comprising:
writing the corrected information to the memory in a write corrected information stage of the processor pipeline.

Evidence Appendix

No Exhibit.

Related Proceedings Appendix

No related proceedings.